A projection-based data partitioning method for distributed tomographic reconstruction

Jan-Willem Buurlage* Rob H. Bisseling† Willem Jan Palenstijn*
K. Joost Batenburg*‡

Abstract

Tomography is a non-destructive technique for imaging the interior of a 3D object. We present an efficient data partitioning strategy for distributed tomographic reconstruction algorithms. Our novel partitioning method is a refinement of the previously published GRCB algorithm. Instead of taking as input a discrete set of lines corresponding to source–pixel pairs, the introduced algorithm works directly on the (cone-shaped) projections. We introduce a geometric characterization of the communication volume, as well as a continuous model for load-balancing based on the varying line densities throughout the object volume. The resulting algorithm is orders of magnitude faster than the original algorithm while producing partitionings of similar quality. We introduce a novel communication data structure that can efficiently represent the communication metadata. An implementation on top of Bulk and the ASTRA toolbox is discussed. We provide experimental results of our method for various commonly used acquisition geometries. We achieve a speedup of 2.8× compared to ASTRA-MPI when using 32 GPUs to reconstruct an image for a circular-cone beam acquisition geometry.

1 Introduction

With tomographic techniques, the interior of an object can be imaged without destroying the object, which makes tomography an important tool in medicine, industry, and science. Using a beam of penetrating radiation, consisting of, e.g., photons or electrons, two-dimensional projections of an object are acquired. These projections can be related to integrals of some volumetric property of the object, such as its density. *Computed Tomography* (CT) is a technique to retrieve a 3D profile of this property from the measured projection images [4, 11].

A tomographic experiment is performed using a *source* that emits the penetrating radiation, and a two-dimensional *detector* that captures the projection images. A finite number of projections are taken of the object. In this article, we will consider *point sources*, and rectangular *flat panel detectors*. This means that each projection corresponds to a cone, with at its base the detector, and at its apex the source.

Two important operations in CT algorithms are the *forward projection* and the *backprojection*. A forward projection operation is a linear transformation that models the physical experiment. It takes a discretized representation of the object, and outputs the two-dimensional projections of the object. The *backprojection* operator is the adjoint of the forward projection operator. Various models can be used for this linear transformation [13, 14, 18].

There are a broad variety of reconstruction algorithms for CT. An important subset of these algorithms uses forward projection and backprojection operations, and these operations typically dominate their runtime costs. Our focus in this article is on reconstruction methods that alternate between forward projection and backprojection operations, with optionally some in-between operations in the image or measurement domain. These include SIRT [9], Krylov methods such as CGLS [10], ML-EM [12], and methods originating from convex optimization such as FISTA [1] and Chambolle–Pock [7].

The computational cost of these reconstruction methods grows superlinearly with respect to the input data. The size of typical tomographic data sets is rapidly increasing, due to advances in hardware and increased interest in multi-modal imaging, imaging of dynamic systems, and adaptive acquisition. Large data sets of many GBs in size are increasingly common, and for these data sets even optimized GPU implementations do not always suffice to keep the computational costs manageable. This motivates the move to large distributed-memory compute clusters, to keep reconstruction times reasonable.

*Centrum Wiskunde & Informatica, PO Box 94079, 1090 GB Amsterdam, the Netherlands.
†Mathematical Institute, Utrecht University, PO Box 80010, 3508 TA Utrecht, the Netherlands.
‡Mathematical Institute, Leiden University, PO Box 9512, 2300 RA Leiden, the Netherlands.

When performing projection operations on a distributed-memory system, communication is the main bottleneck for algorithms that make use of alternating forward projection and backprojection operations. The data partitioning method presented in this article concerns itself with minimizing the required communication, without changing the overall structure of the underlying algorithms, for an arbitrary acquisition geometry, i.e., a set of source and detector positions. It is a refinement of the previously published GRCB partitioning algorithm [3].

While the GRCB method has a good time complexity compared to, e.g., the projection operations, it is still too slow to apply in real time. This limits its applicability in various situations, such as adaptive acquisition where the user may want to zoom in on a region-of-interest after initial inspection, or in cases where the acquisition geometry simply changes from scan to scan, because the user changes, e.g., the source-to-object distance, or the source-to-detector distance.

This article is structured as follows. In section 2, we discuss how to model tomographic reconstruction as a linear inverse problem, discuss an associated partitioning problem, and summarize the original GRCB partitioning method. In section 3, we introduce a geometric characterization of the partitioning problem, and use this to develop a more efficient partitioning algorithm. In section 4, we introduce a memory-efficient data structure that stores communication metadata. In section 5, we give the results of our numerical experiments. Finally, in section 6, we present our conclusions.

2 Background

Tomographic reconstruction. Tomographic reconstruction can be modeled as a linear system of equations. The physical model is discretized in order to obtain a matrix $W \in \mathbb{R}^{m \times n}$, that maps a discretized representation $\mathbf{x} \in \mathbb{R}^n$ of the object (the *image*), to a vector of measurements $\mathbf{b} \in \mathbb{R}^m$. A component x_j corresponds to the jth voxel in the volume. A component b_i corresponds to a measurement for the ith ray, between a source point and a pixel on the detector. The reconstruction problem in tomography is a linear inverse problem of the form: given W and $\mathbf{b}$, find $\mathbf{x}$ such that:

$$W\mathbf{x} \approx \mathbf{b}.$$

In order to construct the *system matrix* W we introduce two concepts: the *acquisition geometry*, and the *object volume*. The acquisition geometry is a set of line segments in three-dimensional space. For each projection, where the radiation source and detector positions are fixed, each *detector element* on the detector (corresponding to a pixel in the projection image), is the end point of a line segment that starts at the position of the source. The imaged object is represented as a discretized volume of *voxels*. Each voxel corresponds to a small cube, and the associated value x_j corresponds to some volumetric property of the object, such as its density, at the location of the voxel.

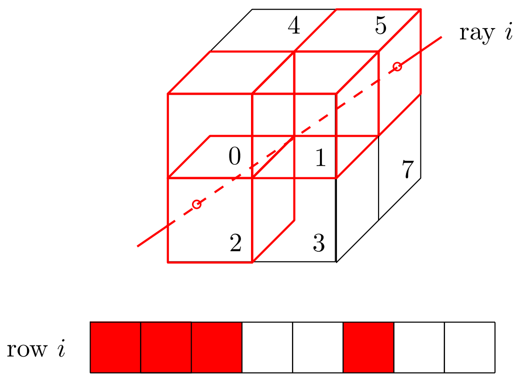

Figure 1: Constructing a row of the system matrix W. The object volume is discretized into $2 \times 2 \times 2$ voxels, and a ray from the acquisition geometry intersects this volume. Here, it passes through four of the numbered voxels, the ones marked red, leading to four nonzeros in the corresponding matrix row.

We do not consider parallel-beam geometries, where conceptually the source is infinitely far away, as they are usually easier to partition. However, the method we present should generalize to those geometries as well.

Each row of the matrix W corresponds to a line segment in the acquisition geometry. Each column of W corresponds to a voxel of the object volume. We assume that the matrix elements W_{ij} are given by the length of the intersection of the ith line with the jth voxel. Note that W is sparse, as each line will only intersect a relatively small collection of voxels. This construction is illustrated in figure 1.

The forward projection and backprojection operations that are crucial for many reconstruction algorithms, correspond to sparse matrix–vector (SpMV) products with W and W^T, respectively.

Parallel execution of projection operations. When the sparse matrix–vector products $\mathbf{y} = W\mathbf{x}$ and $\mathbf{x} = W^T\mathbf{y}$ are executed on a distributed-memory system that consists of p nodes (or *processing elements*, or simply *processors*), communication between the nodes is the single most important consideration for the computational efficiency. The relevant data are the nonzeros of the matrix W, the components of the image $\mathbf{x}$, and the components of the measurements $\mathbf{y}$. For each of these three types of data, a suitable p-way partitioning has to be chosen.

The sth part of the data, is assigned to the sth processor. The three types of data: the image, measurements, and nonzeros, correspond to three ways of partitioning the underlying sparse matrix. An image partitioning implies a *column partitioning* of the matrix, a measurement partitioning implies a *row partitioning* of the matrix, and finally a nonzero-based partitioning gives a *2D partitioning* of the matrix.

Communication occurs because different processors depend on the same data. Each nonzero W_{ij} corresponds to two floating-point operations (flops), as it has to be multiplied with image component x_j and the result of this multiplication occurs in the sum for the measurement component $y_i = \sum_{W_{ij} \neq 0} W_{ij} x_j$. In other words, a nonzero element W_{ij} couples the jth component of $\mathbf{x}$ and the ith component of $\mathbf{y}$. Communication is usually unavoidable if one requires a balanced partitioning where each part is of roughly equal size, but by choosing a suitable partitioning the total *communication volume*, i.e., the number of data words sent, can be reduced significantly. The components of the vectors $\mathbf{x}$ and $\mathbf{y}$ must also be assigned to a processor, without any restriction. In that case, the parallel algorithm will have four phases: (i) a *scatter* phase where each component x_j is communicated to the processors that need it; (ii) a local computation of products $W_{ij} x_j$ followed by an addition of products for the same row i; (iii) a *gather* phase where the contributions to each component y_i are communicated to the owner of the component; (iv) a local addition of the received contributions for each component y_i.

Partitioning for SpMVs is a well-studied problem in combinatorial scientific computing. The underlying structure is modeled as a hypergraph, where common models include *row-net* and *column-net* [6], *medium-grain* [16], and *fine-grain* [5]. Partitioning methods aim to find a balanced partitioning of the vertices of the model hypergraph, that minimizes the total communication volume and in certain cases also the number of messages sent.

The system matrix for a tomographic reconstruction problem is sparse and consists of $\mathcal{O}\left(mn^{1/3}\right)$ nonzeros, and common values for m and n are 10^9 or even higher. This corresponds to many terabytes of data, which means that the matrix cannot be stored explicitly for the desired high resolutions, even when employing a sparse data structure, and that the forward and backprojection must be implemented in a matrix-free manner. This also means that it is not at all clear how SpMV partitioning approaches can be applied. Instead, we consider the underlying geometry of the problem.

Tomographic partitioning problem. In tomographic reconstruction, a cuboid region $V \subset \mathbb{R}^3$ called the *object volume* is defined. The sample being scanned is completely contained in V, and after discretizing the object volume into $n = n_x \times n_y \times n_z$ voxels, the sample can be represented using an image $\mathbf{x}$ with one component for each voxel. For distributed-memory tomographic reconstruction, we choose to find a suitable partitioning of the object volume V, which after discretizing gives a partitioning of the image $\mathbf{x}$, corresponding to a column partitioning of the matrix. The relevant part of W can be generated locally on each processor. Only contributing partial sums for the projection data have to be communicated during the projection operations.

The quality of a partitioning is judged on two grounds: the amount of communication it induces, and whether or not the parts are roughly equal in terms of computational cost.

Instead of considering the nonzeros, we can look at the problem geometrically. A tomographic measurement consists of a number of projections, and for each projection we consider the line segments from the source position to each pixel on the detector. This defines a set of line segments $\mathcal{G}$ that we call the *acquisition geometry*. Communication is required for each line in the acquisition geometry that travels through multiple parts of the image volume. The number of parts a line ℓ crosses for a partitioning π is denoted by $\lambda_\ell(\pi)$. Since we can designate one of the parts as the *owner* of the line, we have for the communication volume:

$$\Lambda(\pi) = \sum_{\ell \in \mathcal{G}} (\lambda_\ell(\pi) - 1).$$

For a good partitioning π, this value will be manageably small.

The computational cost of a part is modeled as the number of flops it has to perform in a projection operation. Each voxel is involved in twice as many flops as there are lines $\ell \in \mathcal{G}$ crossing the voxel. For the jth voxel, we write $\omega(j)$ for the number of lines crossing the voxel. The total computational weight of the sth part is then given by:

$$T^{(s)} = \sum_{j\,:\,x_j \in V_s} \omega(j).$$

Here, the notation $x_j \in V_s$ indicates that the voxel x_j is assigned to the sth part after discretizing. For a good partitioning, the following *load imbalance* ϵ should be kept small:

$$\epsilon(\pi) = \max_{0 \leq s < p} \frac{T^{(s)}}{T_{\text{avg}}} - 1,$$

where $T_{\text{avg}} = \sum_s T^{(s)}/p$. We can summarize the tomographic partitioning problem as follows.

Definition 2.1. Let $\mathcal{G}$ be an acquisition geometry, V the object volume, $\epsilon_{\max}$ the maximum allowed load imbalance, and p the number of processors. Let Π denote the set of all p-way volume partitionings. The tomographic partitioning problem is the following optimization problem:

$$\begin{aligned} \text{minimize}_{\pi \in \Pi} \quad & \Lambda(\pi) \\ \text{subject to} \quad & \epsilon(\pi) \leq \epsilon_{\max}. \end{aligned}$$

Geometric recursive coordinate bisectioning. The GRCB algorithm only looks at partitionings π that are obtained by recursive coordinate bisectioning. That is to say, the volume is recursively split $p-1$ times, each time along one of the axes. Axis-aligned cuboid partitionings such as the ones obtained by GRCB are convenient in practice, and can be expected to give reasonably good results. Because the communication volume is additive (see theorem 2 in [3]), bisectioning can be done independently for each part, which is why we can obtain a good partitioning for any number of processors by recursively splitting in two.

The splitting subroutine of GRCB that performs the bisectioning, is based on a plane sweep. We are able to identify which splitting plane, among all the possible axis-aligned ones, is able to best limit the communication volume by directly considering all the lines in the acquisition geometry $\mathcal{G}$.

3 A new projection-based partitioning method

The GRCB algorithm uses a discrete model for the acquisition geometry, explicitly considering a set of rays. While this leads to an exact representation of computation load (in flops) and communication volume (in data words), it does mean that the input data sizes for the partitioning method are large.

Here, we take a different approach and use a continuous model for the acquisition geometry, communication volume, and computational load. For fine enough resolutions, we expect the discretization error incurred by this model to be small. Instead of minimizing the communication volume subject to a load balance constraint, we now aim to minimize the communication volume and the load imbalance simultaneously by generating a candidate split for each of the three coordinate axes on the basis of load balance, and among these candidate splits choose the one that realizes the lowest communication volume.

As before, the object volume is a cuboid $V = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_2] \subset \mathbb{R}^3$ that we want to partition into p parts. We limit ourselves to partitionings obtained by recursive bisectioning. In this article, the faces of a cuboid are considered part of the cuboid.

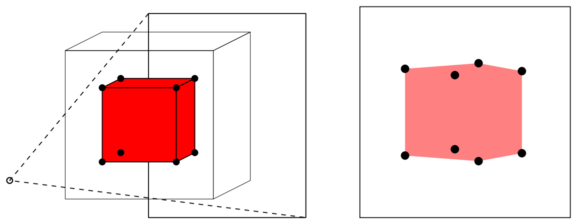

Figure 2: The shadow of a part with respect to the point source defines the region on the detector for which line segments cross the part. Here, the part and its shadow are shown in red. The shadow can be computed by projecting the eight vertices of the cuboid on the detector, and then taking their convex hull.

The acquisition geometry is modeled as a set P of cone-shaped projections p_k. Each projection p_k can be described by a source–detector pair. The point-source is at position $\mathbf{s}_k \in \mathbb{R}^3$. The detector is a rectangular region $D_k \subset \mathbb{R}^3$. The cone with base D_k and apex $\mathbf{s}_k$ defines the projection p_k.

3.1 Communication volume.

We consider the effect that one of the projections $p_k = (\mathbf{s}_k, D_k)$ has on the communication volume. In the discrete model, the volume depends on the resolution on the detector, i.e., the *shape* in pixels of the detector D_k, e.g., 2000×2000. For each detector pixel with center $\mathbf{d}_k^{(i)}$, we consider the line segment ℓ from $\mathbf{s}_k$ to $\mathbf{d}_k^{(i)}$. The number of cuts in ℓ, which is the number of additional parts of the object volume that it intersects, is the contribution in number of data words to the communication volume. Note that we determine a single p-way partitioning of the object volume for the set of all rays from all projections.

We describe here a new approach that works directly on the cones defined by the projections, rather than the individual pixels. It is therefore independent of the detector discretization, and this greatly reduces the size of the input data to the partitioning algorithm.

We exploit the fact that line segments corresponding to neighbouring pixels often cross the same parts. We want to group rays by identifying pixels in a region of the detector for which the corresponding line segments all cross exactly the same parts. The key observation that makes this possible is that a region of the detector for which the line segments cross a given part of the object volume, corresponds to the shadow of that part onto the detector. This is illustrated in figure 2.

The communication volume in our continuous model is estimated in the following way. We consider a candidate split into two parts. Strategies to generate these candidate splits are discussed later. This split happens along one of the axes of the object volume, at

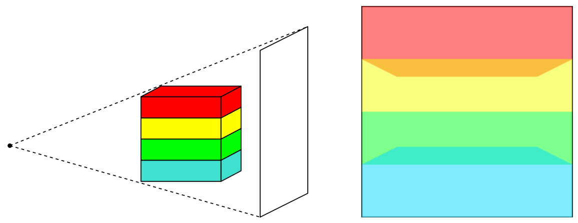
Figure 3: Where shadows of a part overlap, line segments in that region cross multiple parts.

a given location. The split induces two subvolumes, one to the *left* of the splitting plane, and one to the *right*. To identify the region on the detector for which the line segments cross both parts, we forward project the vertices of these subvolumes onto the detector. The shadow of each subvolume can be found by taking the convex hull of its projected vertices. The *area* of the intersection of the two shadows is proportional to the number of line segments crossing both parts for any fine enough discretization of the detector. We compute this for each projection in P in order to find the total communication volume.

Algorithm 1 Computing the communication volume for a given split. Here, M is a magnification value, relating the detector size to the object volume size, and V_L and V_R are cuboids corresponding to the volumes to the left and to the right of the candidate splitting plane.

```
Subroutine: COMMUNICATIONVOLUME
Input: $V_L$, $V_R$, $P$
Output: $\Lambda$

$\Lambda \leftarrow 0$
for all $p_k \in P$ do
    $S_L \leftarrow \text{CONVEXHULL}(\text{PROJECT}(p_k, \text{VERTICES}(V_L)))$
    $S_R \leftarrow \text{CONVEXHULL}(\text{PROJECT}(p_k, \text{VERTICES}(V_R)))$
    $\Lambda \leftarrow \Lambda + \text{AREA}(S_L \cap S_R)$

if consider gradient then
    $\Lambda \leftarrow \Lambda + M \times \text{AREA}(V_L \cap V_R)$
```

A subroutine for computing the communication volume for any candidate split of the volume V into a left part V_L and a right part V_R is given in algorithm 1, optionally taking into account volume for a gradient-based regularizer as discussed in section 3.3.

Because the communication volume is additive, we can split the volume recursively. After $p-1$ splits, we have obtained a partitioning into p parts. The interplay between shadow intersections and communication volume for a fixed projection is illustrated in figure 3.

3.2 Load balance.

We next discuss generating a set of candidate splits that we want to evaluate. These candidate splits should divide the object volume into parts with roughly equal computational weight, and among that set we choose the one that induces the least amount of communication.

Modeling the computational weight in our continuous setting does not appear to be as straightforward as for the communication volume. Recall that the computational weight of a voxel is defined as the number of lines intersecting it. We no longer have an explicit set of lines nor of voxels, but regardless of the discretization we have that the line density in the volume for a given projection decreases as $1/r^2$ where r is the distance to the source. The computational weight of a part V_s should therefore be proportional to the integral:

$$\sum_{k=1}^{|P|} \int_{V_s} \frac{1}{||\mathbf{x} - \mathbf{s}_k||_2^2} d\mathbf{x}. \tag{3.1}$$

If we want to split along, say, the x axis, into two parts with equal computational weight, then we want to find $c \in [x_1, x_2]$ so that

$$\int_{x_1}^{c} \int_{y_1}^{y_2} \int_{z_1}^{z_2} \sum_{k=1}^{|P|} \frac{1}{||\mathbf{x} - \mathbf{s}_k||_2^2} \, dz \, dy \, dx = \int_{c}^{x_2} \int_{y_1}^{y_2} \int_{z_1}^{z_2} \sum_{k=1}^{|P|} \frac{1}{||\mathbf{x} - \mathbf{s}_k||_2^2} \, dz \, dy \, dx.$$

The volume integral for a rectangular volume $V = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_2]$ can be written as the following 2D integral:

$$\int_{x_1}^{x_2} \int_{y_1}^{y_2} \sum_{k=1}^{|P|} \left(\frac{1}{a_k(x,y)} \left(\arctan\left(\frac{z_2 - s_{k,z}}{a_k(x,y)} \right) - \arctan\left(\frac{z_1 - s_{k,z}}{a_k(x,y)} \right) \right) \right) dy \, dx, \tag{3.2}$$

where

$$a_k(x,y) = \sqrt{(x - s_{k,x})^2 + (y - s_{k,y})^2}.$$

This is, of course, more efficient to solve numerically compared to the original three-dimensional problem.

For finding c, we use the following strategy. We take N samples in the volume V. Next, we choose c such that

$$(c - x_1)\bar{f}_L = (x_2 - c)\bar{f}_R,$$

where $\bar{f}_L$ is the average of the integrand in (3.1), or the more efficient variant in (3.2), for samples with an x-coordinate smaller than c, and $\bar{f}_R$ for the remaining samples. We find the optimal c by sorting the N samples by their x-coordinate, and performing a linear scan while updating the averages to the left and right of c. It is possible to decide on the number of samples N dynamically, by updating c for each new sample, and taking samples until the optimal value for c converges.

A difficulty is introduced for acquisition geometries where, because of a limited detector size, or a source that is close to the object, the object volume is not contained in the cones defined by the projections. In these cases, we want to integrate over the intersection of the cone and the volume. This can be easily realized by rejecting samples for a projection p_k if the sample projects to a point outside of the detector. For these acquisition geometries, we cannot employ the analytical reduction from 3D to 2D shown in (3.2).

As an alternative to approximating the above integrals numerically, we can employ a simpler strategy to identify valid candidate splits. We still consider each axis in turn. If we split *in the middle* along a given axis, we end up with two parts that are equal in volume and should thus have the same number of voxels (up to discretization errors). If the number of lines intersecting a voxel is more or less constant throughout the volume, the number of voxels is one way to achieve a reasonable load balance.

Solving the numerical integraton problem, or using the splitting in the middle strategy (which we will refer to as MIDWAY in our experiments), both result in three candidate splits, one for each axis. Out of these three candidate splits, the best one is chosen each time, based on communication volume.

3.3 Image gradient computations. Image gradient computations form an optional component of a number of reconstruction methods. Prior information on the object, such as the object being piecewise constant, or being smooth, can be incorporated as a penalty term involving the norm of the image gradient. In these cases, tomographic reconstruction is performed by solving a regularized least-squares problem:

$$\operatorname{argmin}_{\mathbf{x}\in\mathbb{R}^n} ||\mathbf{b} - W\mathbf{x}||_2^2 + \lambda||\nabla\mathbf{x}||_1. \tag{3.3}$$

To perform (discrete) gradient computations, each processor requires the value of the neighbouring voxels to each of its voxels. This means that values for voxels that lie next to the splitting plane have to be obtained from a remote processor. In previous work, this communication cost was ignored in the partitioning algorithm. However, it is straightforward to include this as a term in the communication volume, by considering the area of the splitting plane in addition to the area of the shadow intersections.

Both the area of the splitting plane, and the area of the shadow intersections on the detector are proportional to their respective communication weights, but by a different coefficient. Therefore, the areas should be normalized, so that they can be compared to each other. The discretization on the detector should take into account the total area of the detector, and the discretization of the object volume should in turn take into account its total volume.

In particular, discretization is commonly chosen so that if a voxel in the volume has a cross-section of area X, then the area of its shadow Y corresponds roughly to the size of a detector pixel. We will use the magnification value $M = Y/X$ to relate the communication volumes due to gradient computations and due to an SpMV.

In our new algorithm, this communication volume for gradient computations is optionally taken into account. When splitting a part that is elongated in some direction, the cross-section (area of the splitting plane) will depend on the axis chosen, and this can influence the resulting partitioning.

4 Communication data structures

In this subsection, we will discuss how to use the partitionings efficiently in practice. The partitionings aim to minimize the communication volume, while evenly sharing the work among the processors. However, performing the communication requires storing information on what gets sent where during the execution.

The iterative algorithms that are the focus of this work, perform alternating forward projection and backprojection operations. During a forward projection, that is the calculation of $\mathbf{y} = W\mathbf{x}$, *contributions* to the components of $\mathbf{y}$ are computed by the processors whose part of the object volume is crossed by the line segment corresponding to that component. Therefore, each component of $\mathbf{y}$ has one or more *contributing processors*. One of these contributing processors is designated as the owner of the component. The owner computes the sum of the contributions. Before a backprojection $\mathbf{x} = W^T\mathbf{y}$, this sum is distributed to the group of contributing processors. With this *gather–scatter* setup the modeled communication volume is realized in practice.

The *communication data structure* contains information on the sets of contributing processors for each component. This information has to be stored so that the gather and scatter operations can be executed efficiently in every iteration.

A straightforward way to build the communication data structures, is to compute and store for each individual component its set of contributing processors, and to designate one of them as an owner (e.g., at random or through a round-robin scheme). However, this will severely increase the memory use, since the size of the communication data structures for a realistic number of processors will be bigger than the projection data itself. This is because the metadata, that identifies what is being communicated, is associated with every individual component.

To remedy this problem, we again exploit the fact that line segments corresponding to neighbouring pixels on the detector often cross the same parts. In particular, we would like to find the regions of pixels of projection images that have the same set of contributing processors. This can be realized by looking at arrangements induced by the shadows of each part of the partitioning.

An *arrangement* is a subdivision of the plane into a collection of labeled regions, or *faces*. In our case, we are interested in subdivisions of the detector plane, and the labels (or *tags*) are the sets of contributing processors for the face.

We consider each projection separately. Every processor shadow defines an arrangement of the rectangle of the detector containing two faces: the shadow of the part, and its complement. The p arrangements can be merged efficiently, as described in section 2.3 of the textbook by de Berg et al. [8]. The resulting overlay arrangement has faces defined by the intersections of the faces in the original arrangements, and the tags can be combined arbitrarily. In our case, the faces in the original arrangements have a single contributing processor corresponding to a tag that is a list with one element. We start with an empty arrangement, and iteratively merge in the arrangements for each processor. When new faces are constructed during the MERGE subroutine, the lists of contributors of the original faces are concatenated. After merging together the p arrangements, the resulting overlay structure defines a number of faces, and each of these faces has an associated set of contributing processors as defined by its tag. We summarize this method in algorithm 2.

Our novel communication data structure is thus a subdivision of the detector into a set of faces, with an associated tag for each face listing the contributing processors for that region. For a visual example, see figure 4. We then proceed to rasterize these faces, leading for each face to a collection of scanlines. A *scanline* is a consecutive set of pixels of a row on the detector. We use this collection of scanlines in the final algorithm for performing the communication during an SpMV operation. This novel approach not only drastically reduces the size of the communication data structures, but also allows to perform aggregate reads from GPU memory.

Algorithm 2 Finding the overlay for the communication structure for a given projection p_k. Here, $[s]$ is a list with a single element s.

```
Subroutine: FINDFACES
Input: $\pi = \{V_s\}, p_k$
Output: OVERLAY

OVERLAY ← EMPTYARRANGEMENT
for $0 \leq s < p$ do
    CORNERS$_s$ ← PROJECT($p_k$, VERTICES($V_s$))
    SHADOW$_s$ ← CONVEXHULL(CORNERS$_s$)
    ARRANGEMENT$_s$ ← FROMFACETAG(SHADOW$_s$, $[s]$)
    MERGE(OVERLAY, ARRANGEMENT$_s$, CONCATENATE)
```

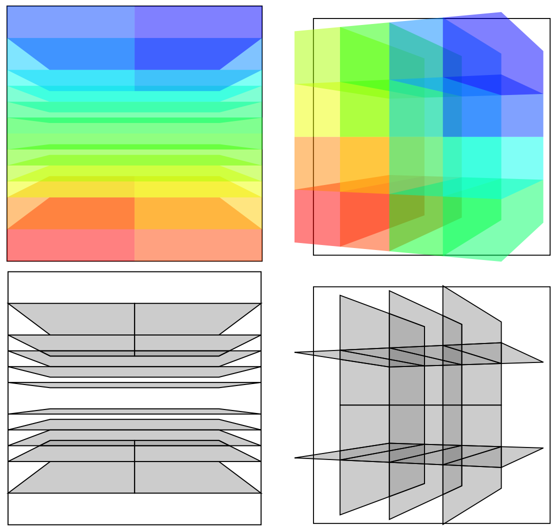

Figure 4: Example of the overlay structure for a single projection of the CCB$_w$ (left), and LAM$_w$ (right) geometries (see section 5). Note that the shadows of a part might partially fall outside of the detector. On the top row, the shadows of the coloured parts are given. On the bottom row, the overlay structure is shown. In the overlay, a darker gray indicates a larger set of contributing processors.

5 Numerical experiments

We consider four categories of acquisition geometries for our numerical experiments.

- CCB. *Circular cone-beam.* The source and detector move in a circular trajectory around the object. This is the typical geometry for laboratory CT machines. We distinguish between CCB$_n$ where the cone has a narrow angle, and the source is relatively far away, and CCB$_w$ with a wide angle, and the source is close to the volume.

- HCB. *Helical cone-beam.* The source and detector move in a helical trajectory around the object. This is similar to CCB, but in addition to the circular movement, the source and detector also move along the orthogonal direction. This is a common acquisition geometry in medical CT.
- LAM. *Laminography.* The source and detector both move along their own circular trajectory, but these trajectories are on opposite sides of the volumes, typically perpendicular to one of the axes of the object volume. This geometry can be used to image flat objects. We distinguish between LAM_w with circular trajectories with a large radius, and LAM_n with a small radius.
- TSYN. *Tomosynthesis.* A static detector is placed under the object, while the source moves along a limited-angle arc above the object. This geometry is used, e.g., for breast cancer screening and airport security.

Partitioning results. Here, we compare two methods for load balancing that were discussed in section 3.2, MIDWAY where we split the volume into two parts of equal volume, and SAMPLING where we take a fixed number $N = 100\,000$ of sample points for which we evaluate the integrand in (3.1), and then perform a linear scan to find the optimal splitting point. We compare the results for these methods with the original GRCB partitioning method. In practice, these partitionings are of interest for multi-GPU clusters consisting of up to $p = 64$ GPUs. We therefore consider three processor counts, 16, 32, and 64. The partitioning statistics such as communication volume and load imbalance are evaluated on volumes consisting of 256^3 voxels, which is fine enough to obtain accurate statistics. We show some of the partitionings visually in figure 5.

In table 1, we show the communication volume Λ, load imbalance ϵ, number of messages μ (i.e., the number of processor pairs that perform the communication), and partitioning time T. Note that we do not optimize for the number of messages explicitly. First, we observe that there are no large discrepancies in the communication volume between the three different methods. For MIDWAY, the partitioning time is low (between 100 ms–600 ms), but the load imbalance can be up to 0.34 for the geometries considered. The number of messages is somewhat lower compared to the other partitioning algorithms, since the parts are automatically aligned because of the fixed split points, which is beneficial for the number of messages. The maximum number of messages is $\mu_{\max} = p(p-1)$, and we note that the number μ achieved is often a significant fraction of $\mu_{\max}$. This attests to the difficulty of avoiding communication in tomography, caused by rays crossing the object in many directions. We see that the SAMPLING method based on our continuous formulation of the load balance is able to achieve a reasonable load balance. Only in two cases it is slightly higher than the maximum load imbalance (0.05) that was imposed for GRCB. The runtime of the partitioning algorithm is up to 100× less than the runtime of GRCB, while the resulting partitionings have similar quality.

In table 2, we consider the communication volume for regularized reconstruction methods that solve (3.3). We do this by explicitly considering communication because of image gradient computations during the par-

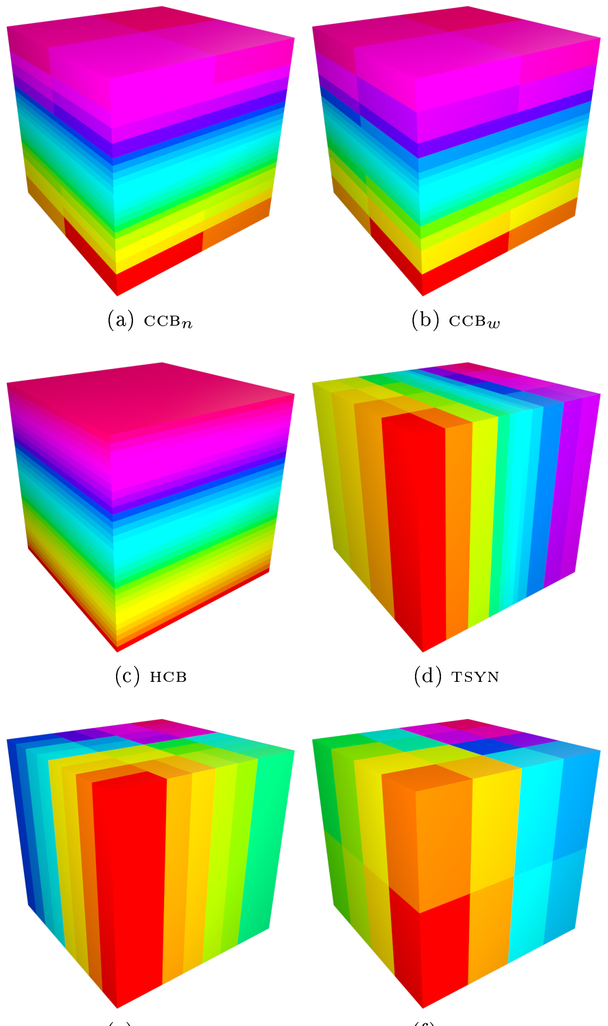
(a) CCB_n (b) CCB_w (c) HCB (d) TSYN (e) LAM_n (f) LAM_w

Figure 5: Resulting partitionings for the circular cone-beam (CCB), helical cone-beam (HCB), tomosynthesis (TSYN), and laminography (LAM) acquisition geometries. The results shown are for $p = 32$ processors using the MIDWAY load balancing strategy.

		MIDWAY				SAMPLING				GRCB			
p	$\mathcal{G}$	Λ	ϵ	μ	T	Λ	ϵ	μ	T	Λ	ϵ	μ	T
16	CCB$_n$	0.72	0.00	44	0.15	0.72	0.00	40	6.72	0.72	0.00	44	242.54
	CCB$_w$	1.26	0.01	64	0.09	1.26	0.04	64	4.95	1.26	0.03	64	274.98
	HCB	1.14	0.28	84	0.16	1.18	0.04	96	4.76	1.18	0.05	96	203.48
	LAM$_n$	0.92	0.00	84	0.15	0.92	0.04	140	7.28	0.91	0.05	120	240.11
	LAM$_w$	1.61	0.00	180	0.14	1.61	0.03	180	7.04	1.61	0.05	180	296.20
	TSYN	0.72	0.10	76	0.15	0.73	0.03	76	2.22	0.72	0.05	76	210.32
32	CCB$_n$	1.28	0.00	180	0.32	1.28	0.00	172	4.75	1.28	0.04	180	273.62
	CCB$_w$	1.90	0.01	272	0.31	1.90	0.04	272	4.74	1.90	0.04	272	350.76
	HCB	1.91	0.33	328	0.33	1.96	0.04	350	4.92	1.98	0.05	368	296.13
	LAM$_n$	1.53	0.01	340	0.24	1.53	0.05	446	7.23	1.53	0.05	394	330.22
	LAM$_w$	2.61	0.17	552	0.19	2.66	0.03	552	7.23	2.65	0.05	552	347.40
	TSYN	1.19	0.10	284	0.19	1.19	0.05	272	2.43	1.19	0.05	284	253.28
64	CCB$_n$	1.92	0.04	640	0.38	1.92	0.04	648	5.43	1.92	0.05	640	324.07
	CCB$_w$	2.86	0.01	1040	0.37	2.85	0.04	1040	5.14	2.85	0.05	1040	449.93
	HCB	2.74	0.34	1052	0.40	2.80	0.05	1170	5.25	2.79	0.05	1150	400.16
	LAM$_n$	2.21	0.01	1268	0.37	2.20	0.06	1412	10.17	2.31	0.04	1346	480.77
	LAM$_w$	3.68	0.17	1978	0.60	3.74	0.06	2048	10.29	3.73	0.05	2020	536.73
	TSYN	1.79	0.12	936	0.40	1.80	0.05	928	4.09	1.80	0.05	948	246.44

Table 1: Partitioning statistics. We compare the communication volume Λ, given in multiples of 10^7, the load imbalance ϵ, the number of messages μ and the partitioning time T in seconds for various combinations of processor count p and acquisition geometry $\mathcal{G}$.

titioning method, or ignoring this cost, as explained in section 3.3. The effect is limited because the communication due to image gradient computations is relatively small, as especially for larger processor counts the total communication volume is dominated by that of the SpMV step. However, it improves the overall communication in some cases, up to 3% for CCB$_n$, which is an acquisition geometry with relatively low communication volume Λ for the SpMVs.

Performance measurements. We have implemented an extension to the open-source ASTRA tomography toolbox that allows tomographic reconstruction algorithms to run on distributed-memory GPU clusters. This extension is called *Pleiades*, after the famous open star cluster. The ASTRA toolbox [17] has highly optimized GPU implementations of projection operators, which we use for the local forward projection and backprojection operations. Our extension uses Bulk [2] to realize the communication between nodes. Bulk is a modern C++ library for bulk-synchronous parallel programs. It simplifies the implementation of communication logic significantly compared to, e.g., BSPlib or MPI.

Our extension is an improvement over a previously published extension to the ASTRA toolbox based on MPI [15], which we will call ASTRA-MPI. This previous extension uses slab partitionings, where the volume is split up into blocks of consecutive slices along one of the axes. This makes it suitable only for circular cone-beam geometries.

In contrast, our distributed-memory extension to the ASTRA tomography toolbox is flexible with respect to the acquisition geometry and the used data partitioning, which we achieved by implementing the techniques outlined in this article. We compare the performance of Pleiades to that of ASTRA-MPI for the only acquisition geometries in our set that ASTRA-MPI supports, which are CCB$_n$ and CCB$_w$. In addition, we test the scalability of Pleiades for HCB. Our test consists of three Landweber iterations defined by the update:

$$\mathbf{x} \leftarrow \mathbf{x} + W^T(\mathbf{b} - W\mathbf{x}),$$

which follows the typical structure of an iterative method by alternating forward projection and backprojection operations. In all cases, we take a volume of 2048^3 voxels, and 1024 projections of 2048×2048 pixels.

The performance tests were run on a compute cluster of 8 nodes with a 40 Gbit Mellanox Infiniband connection. Each node has four NVIDIA GeForce GTX TITAN X GPUs, two Intel Xeon E5-2630 v3 CPUs running at 2.40GHz, and 128GB RAM. We use the MIDWAY strategy for Pleiades, partitioning over

p	$\mathcal{G}$	$\frac{\Lambda^G}{\Lambda}$	$\frac{\Lambda^G_{\text{reg}}}{\Lambda_{\text{reg}}}$	$\frac{\Lambda^G_{\text{total}}}{\Lambda_{\text{total}}}$
16	CCB$_n$	1.00	0.88	0.97
	CCB$_w$	1.00	1.00	1.00
	HCB	1.00	1.00	1.00
	LAM$_n$	1.00	1.00	1.00
	LAM$_w$	1.00	1.00	1.00
	TSYN	1.00	1.00	1.00
32	CCB$_n$	1.00	0.92	0.99
	CCB$_w$	1.00	1.00	1.00
	HCB	1.00	0.94	0.99
	LAM$_n$	1.00	1.00	1.00
	LAM$_w$	1.00	1.00	1.00
	TSYN	1.00	0.92	0.99
64	CCB$_n$	1.00	1.00	1.00
	CCB$_w$	1.00	0.92	0.99
	HCB	1.00	1.00	1.00
	LAM$_n$	1.00	1.00	1.00
	LAM$_w$	1.00	1.00	1.00
	TSYN	1.00	0.95	0.99

Table 2: The relative performance when considering the gradient-based regularization in the communication volume. We compare the communication Λ due to SpMV, the communication Λ_{reg} due to an image gradient computation, as well as the total communication $\Lambda_{\text{total}} = \Lambda + \Lambda_{\text{reg}}$. The fractions given are the communications when explicitly taking into account the gradient communication during the partitioning (marked with a superscript G), divided by the communication when this cost is ignored, both for the SAMPLING method.

the 32 GPUs. Figure 6 shows the results of these measurements. For some acquisition geometries, the amount of available memory made it impossible to run with a low number of GPUs. Our initial implementation of Pleiades supports only $p = 2^q$ processors. We observe that Pleiades is significantly faster than ASTRA-MPI, and Pleiades continues to scale even when using all the available GPUs, unlike ASTRA-MPI which reached a communication bottleneck for CCB$_w$ at around 16 GPUs.

6 Conclusion

We presented a new partitioning method for tomographic reconstruction that can handle arbitrary acquisition geometries. Furthermore, we introduced an efficient data structure for the communication metadata that needs to be stored to use these partitionings in practice. We demonstrated that the method is able to produce partitionings of similar quality to those produced by the previously published GRCB method, but is much faster. Finally, we showed scalability results for

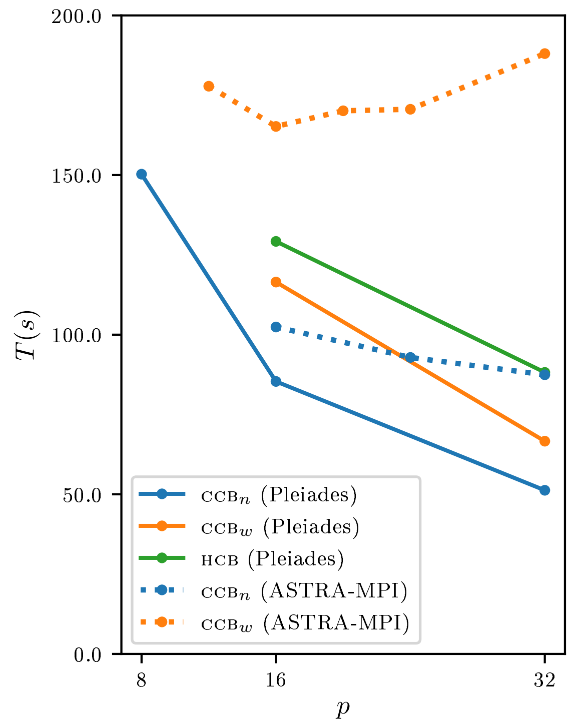

Figure 6: Scaling results of Pleiades versus ASTRA-MPI. Vertically, the runtime in seconds of three consecutive Landweber iterations is shown. Horizontally, we show the number of GPUs that were used.

using these partitionings in practice for a typical reconstruction task. For CCB$_w$ with 32 GPUs we achieved a speedup of 2.8× compared to ASTRA-MPI.

Acknowledgements

Financial support provided by The Netherlands Organisation for Scientific Research (NWO), project number 639.073.506.

References

[1] A. Beck and M. Teboulle, *A fast iterative shrinkage-thresholding algorithm for linear inverse problems*, SIAM Journal on Imaging Sciences, 2 (2009), pp. 183–202.

[2] J. W. Buurlage, T. Bannink, and R. H. Bisseling, *Bulk: a modern C++ interface for bulk-synchronous parallel programs*, in Euro-Par 2018: Parallel Processing, vol. 11014 of Lecture Notes in Computer Science, Springer, 2018, pp. 519–532.

[3] J. W. Buurlage, R. H. Bisseling, and K. J. Batenburg, *A geometric partitioning method for dis-*

tributed tomographic reconstruction, Parallel Computing, 81 (2019), pp. 104–121.

[4] T. M. Buzug, *Introduction to Computed Tomography: From Photon Statistics to Modern Cone-beam CT*, Springer, 2008.

[5] Ü. V. Çatalyürek and C. Aykanat, *A fine-grain hypergraph model for 2D decomposition of sparse matrices*, in Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001.

[6] Ü. V. Çatalyürek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Transactions on Parallel and Distributed Systems, 10 (1999), pp. 673–693.

[7] A. Chambolle and T. Pock, *A first-order primal-dual algorithm for convex problems with applications to imaging*, Journal of Mathematical Imaging and Vision, 40 (2010), pp. 120–145.

[8] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed., 2008.

[9] P. Gilbert, *Iterative methods for the three-dimensional reconstruction of an object from projections*, Journal of Theoretical Biology, 36 (1972), pp. 105–117.

[10] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, Journal of Research of the National Bureau of Standards, 49 (1952), pp. 409–436.

[11] A. C. Kak and M. Slaney, *Principles of Computerized Tomographic Imaging*, Society for Industrial and Applied Mathematics, 2001.

[12] K. Lange, R. Carson, et al., *EM reconstruction algorithms for emission and transmission tomography*, Journal of Computer Assisted Tomography, 8 (1984), pp. 306–16.

[13] Y. Long, J. A. Fessler, and J. M. Balter, *3D forward and back-projection for X-ray CT using separable footprints*, IEEE Transactions on Medical Imaging, 29 (2010), pp. 1839–1850.

[14] B. D. Man and S. Basu, *Distance-driven projection and backprojection in three dimensions*, Physics in Medicine and Biology, 49 (2004), pp. 2463–2475.

[15] W. J. Palenstijn, J. Bédorf, J. Sijbers, and K. J. Batenburg, *A distributed ASTRA toolbox*, Advanced Structural and Chemical Imaging, 2 (2017), p. 19.

[16] D. M. Pelt and R. H. Bisseling, *A medium-grain method for fast 2D bipartitioning of sparse matrices*, in IEEE 28th International Parallel and Distributed Processing Symposium, 2014, pp. 529–539.

[17] W. van Aarle, W. J. Palenstijn, J. Cant, E. Janssens, F. Bleichrodt, A. Dabravolski, J. D. Beenhouwer, K. J. Batenburg, and J. Sijbers, *Fast and flexible X-Ray tomography using the ASTRA toolbox*, Optics Express, 24 (2016), p. 25129.

[18] F. Xu and K. Mueller, *A comparative study of popular interpolation and integration methods for use in computed tomography*, 3rd IEEE International Symposium on Biomedical Imaging: Macro to Nano, (2006), pp. 1252–1255. .